大数据热点技术综述

陈军成[1], 丁治明[1], 高 需[2]
(1. 北京工业大学计算机学院, 北京 100124; 2. 中国科学院软件研究所, 北京 100190)

摘 要: 大数据是当前学术界和工业界关注的热点,从大数据文件系统、大数据分布式存储策略、大数据资源调度以及大数据计算框架等几个方面阐述了当前典型的大数据相关技术及热点技术,并进一步指出下一步可能的研究方向:如何通过语义提高计算效率以及如何高效处理时空大数据.

关键词: 大数据; 数据存储; 计算框架
中图分类号: U 461; TP 308 **文献标志码**: A **文章编号**: 0254 - 0037(2017)03 - 0358 - 10
doi: 10.11936/bjutxb2016090005

Survey of Big Data Hot Techniques

CHEN Juncheng[1], DING Zhiming[1], GAO Xu[2]
(1. College of Computer Science, Beijing University of Technology, Beijing 100124, China;
2. Institute of Software, China Academy of Sciences, Beijing 100190, China)

Abstract: Big data attracts attention of academia and industry. According to the basic features of big data and current challenges, the typical big data's technologies of big data file system were presented. The storage strategy, resource schedule, big data computing framework, and especially hot techniques were distributed. Last, this paper points out two possible research directions, which are improving compute efficiency through data semantic and processing temporal-spatial big data efficiently.

Key words: big data; data storage; computing framework

近年来,大数据迅速成为工业界和学术界争相讨论的热点,甚至引起了国家层面的关注. 美国政府将大数据看作“未来的新石油”,我国政府则在2015年正式发文《促进大数据发展行动纲要》,从国家层面引导大数据相关产业的发展.

根据维基百科的定义,大数据又称为巨量数据、海量数据、大资料等,是指无法通过人工或者计算机,在合理的时间内达到截取、管理、处理并整理成为人类所能解读的形式的信息[1],通常应用于商业模式及趋势的发现与探究、疾病预测、实时交通等领域,特别是在科学研究领域,如脑科学、基因科学、生物工程等. 通常情况下,科学家面对的是海量数据,很难直接发现其中的因果关系,然而,借助大数据相关技术手段,科学家能相对容易地发现其中的关联关系. 这种关联关系可以进一步指引科学家深入探究其中的因果关系.

与传统的数据相比,大数据具有5 V特征,即数据规模庞大(volume)、速度快(velocity)、形态多(variety)、识别困难(veracity)以及价值大但价值密度低(value)等. 大数据系统通常需要解决如何高效存储数据、如何处理瞬间爆发的数据以及如何应对形态各异的结构化、半结构化以及非结构化数据等问题.

针对这些问题,国际巨头 Google、Facebook、

收稿日期: 2016-09-02
基金项目: 国家自然科学基金资助项目(91546111); 北京市博士后基金资助项目(Q6007011201602)
作者简介: 陈军成(1980—), 男, 讲师, 主要从事大数据、软件测试与分析方面的研究, E-mail: juncheng@bjut.edu.cn

Amazon、Microsoft 和 Apache 的开源组织以及国内的百度、阿里巴巴、腾讯(BAT)等均从各行业实际需求出发，提出了大数据相关文件系统、存储技术、大数据分析引擎等. 本文从技术的角度，对大数据文件系统、存储、大数据资源管理与调度、大数据分析引擎等大数据相关技术进行综述，并进一步指出可能的技术发展方向.

1　大数据文件系统

如何存储海量且形态各异的数据是大数据文件系统需要解决的首要问题. 大数据文件系统是大数据的基础. 大数据以分布式的方式存储，如何在分布式系统中分布数据，如何保证分布式系统中的容错，以及如何处理大数据中的冗余，均是大数据文件系统需要解决的问题. 目前，典型的大数据文件系统包括基于存储的分布式文件系统：GFS(Google file system)[2]和 Hadoop[3]，以及基于分布式内存的文件系统(Tachyon[4]). 分布式文件系统利用 RCFile[5]、Parquet[6]等存储格式优化存储，节省存储空间.

1.1　基于存储的分布式文件系统

分布式文件系统(distributed file system)是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连. 分布式文件系统的设计基于客户机/服务器模式.

大数据庞大的数据规模、复杂的数据结构以及可能面对的高频访问，给操作系统中传统的文件系统提出了挑战. 为了解决这类问题，Google 公司于 2003 年发表了 Google 内部研发的 Google 文件系统 GFS[2]. GFS 针对 Web 环境下批量大规模海量数据处理而制定，虽未公开其源码，但依然在学术界和工业界受到广泛关注. 在 GFS 的影响下，Doug Cutting 和 Mike Cafarella 于 2004 年在 Nutch 中实现了 GFS 的开源版本 HDFS(Hadoop distributed file system)[3]，并且在 2006 年 2 月成为 Apache Hadoop 项目的关键组成部分.

HDFS 是为以流式数据访问模式存储超大文件而设计的文件系统，支持超大文件(数百 TB 甚至 PB 级的数据)，以普通硬件为基础，重点支持一次写入、多次读取的场景.

HDFS 架构如图 1 所示. HDFS 采用主(master)/从(slave)架构. 一个 HDFS 集群是由一个 Namenode 和一组 Datanodes 组成. Namenode 是一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问. 集群中的 Datanode 一般是一个节点，负责管理它所在节点上的存储. HDFS 暴露了文件系统的名字空间，用户以文件的形式在上面存储数据. 从内部看，1 个文件被分成 1 个或多个数据块，这些块存储在 1 组 Datanode 上. Namenode 执行文件系统的名字空间操作，如打开、关闭、重命名文件或目录. 它也负责确定数据块到具体 Datanode 节点的映射. Datanode 负责处理文件系统客户端的读写请求. 在 Namenode 的统一调度下进行数据块的创建、删除和复制.

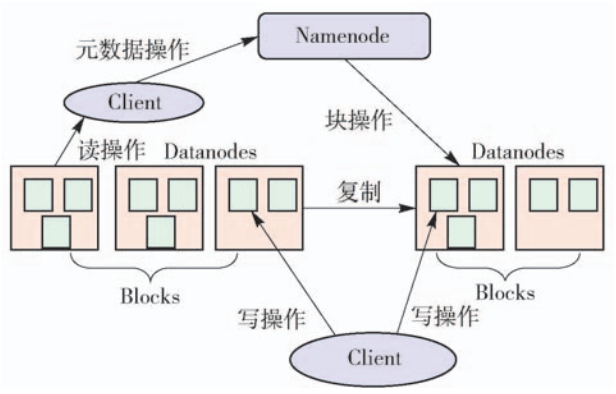

图 1　HDFS 架构图

Fig. 1　HDFS architecture

HDFS 支持在一个大集群中跨机器可靠地存储超大文件. 它将每个文件存储成一系列的数据块，除了最后一个，所有的数据块都是同样大小. 为了容错，文件的所有数据块都有副本. 每个文件的数据块大小和副本系数均可配置，应用程序指定某个文件的副本数目. 副本系数可以在文件创建的时候指定，也可以在之后改变. HDFS 中的文件都是一次性写入，并且严格要求在任何时候只能有一个写入者. Namenode 全权管理数据块的复制，周期性地从集群中的每个 Datanode 接收心跳信号和块状态报告. 接收到心跳信号意味着该 Datanode 节点工作正常. 块状态报告包含了一个该 Datanode 上所有数据块的列表.

HDFS 的主要优势体现在一次写、多次读的应用场景，对于小文件支持开销较大，并且 HDFS 存在单点问题. 针对这些问题，Weil 提出了一套高性能、易扩展的、无单点的分布式文件存储系统 Ceph[7]. Ceph 的主要目标是提供高可扩展性及对象存储、块存储和文件系统的存储机制. Ceph 提供一个单一的存储平台，可以处理所有类型的数据存储(包括对象、块和文件)，其高扩展性可以达到 PB 级，拥有

高容错性和高一致性数据冗余机制.

1.2 基于分布式内存的文件系统

随着实时计算的需求增加,分布式内存计算持续升温,如何将海量数据近实时处理,如何把离线批处理的速度再提升到一个新的高度也是当前研究的重点. 随着固态硬盘(solid state drivers,SSD)等设备的发展,内存吞吐量呈指数增长,而磁盘的吞吐量增长缓慢,将原有计算框架中的磁盘文件替换为内存文件,成为提高效率的优化点.

基于这种考虑,AmpLab 的李浩源等研发了 Tachyon[4]. Tachyon 是一个高容错的分布式内存文件系统,其设计的核心内涵是,要满足当下"低延迟"的数据处理要求. Tachyon 是在内存中处理缓存文件,允许文件以访问内存的速度在集群框架中进行可靠的共享,类似于 Spark[8],Tachyon 的吞吐量比 HDFS 高出 100 倍. Spark 框架虽然也提供了强大的内存计算能力,但其没有提供内存文件的存储管理能力,而 Tachyon 则弥补了 Spark 的不足之处. Tachyon 是架构在最底层的分布式文件存储和上层的各种计算框架之间的一种中间件,其主要职责是将那些不需要存储到 DFS(distributed file system)里的文件放入分布式内存文件系统中共享内存,从而提高效率. 同时可以减少内存冗余、垃圾回收(garbage collect,GC)时间等,Tachyon 在大数据中的层次关系如图 2 所示.

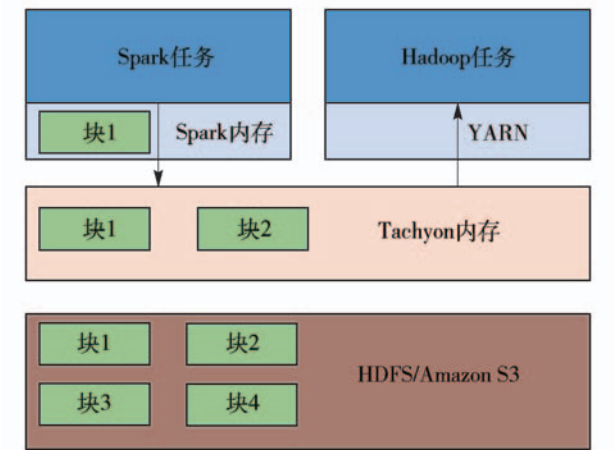

图 2 Tachyon 文件层次架构

Fig. 2 Tachyon file architecture

Tachyon 作为衔接底层文件系统(HDFS、Amazon S3)和上层处理框架的缓存结构,可提高实时分析系统的效率.

在分布式文件系统中,相比而言,HDFS 适合存储大文件,Ceph 适合存储小文件(已被纳入 Linux 内核),而 Tachyon 则是内存文件系统,其地位处于 HDFS 和 Ceph 之上,MapReduce 和 Spark 之下.

1.3 数据文件格式

在文件系统中,数据管理的最大挑战之一是如何管理大数据中的数据冗余. 在传统的数据管理中,存在 2 种数据布局方式,分别是行式存储模式(关系数据库)和列式存储模式.

行式存储模式把不同数据类型的数据列连续、混合地存储在一起,这种情况下要提高数据的压缩率将会非常困难,而数据压缩率不高直接导致使用更多的磁盘存储空间. 列式存储模式则不能保证同一个数据记录的所有字段都在集群中的同一个节点上,因此,重构原表数据记录的时候将在集群网络中造成大量的数据传输,这将产生大笔的性能开销,导致查询的处理速度非常慢.

结合行式存储模式和列式存储模式的优缺点,当前的大数据系统普遍采用混合式存储模式.

RCFile[5]是 Hive[9]推出的一种专门面向列的数据格式. 它遵循"先按列划分,再垂直划分"的设计理念. 在查询过程中,针对它并不关心的列时,它会在输入/输出(input/output,I/O)上跳过这些列. RCFile 在 map 阶段从远端拷贝仍然是拷贝整个数据块,并且拷贝到本地目录后 RCFile 不是真正直接跳过不需要的列.

RCFile 的主要特点包括:1)保证数据记录的所有字段都在集群中同一个节点上;2)集群节点上每一列数据单独压缩,使得 RCFile 的数据压缩率非常高;3)在数据读取的时候只读取需要使用的数据列,不使用的数据列永远不会去读取. 为了进一步降低 NameNode 的负载、提高数据的压缩效率,在 RCFile 的基础上,俄亥俄州立大学的 Huai 等提出了一种基于列式存储的改进版存储格式 ORCFile[10].

嵌套数据是大数据系统中需要处理的常见数据模型,然而 ORCFile 的原生模型对此支持不足. 针对这一问题,在 Dremel[11]的启发下,Twitter 和 Cloudera 合作研发了 Parquet[6]. Parquet 将嵌套结构存储的数据存储成扁平格式,并且在同一个数据文件中保存一行中的所有数据,以确保在同一个节点上处理时每一行的所有列都可用. 另外,Parquet 使用一些自动压缩技术,例如行程编码(run-length encoding,RLE)和字典编码(dictionary encoding,DE),极大地节约了存储空间.

现有大数据文件系统如 Hadoop、GFS 等,充分利用非关系型特征,部分解决了大数据的数据种类

繁多的问题；而 RCFile 等数据文件格式，则对数据进行了充分的压缩，部分解决了大数据的数据规模庞大问题. 在 SSD 逐渐普及的今天，如何充分利用 SSD 构建更加高效的大数据文件系统是一个值得努力的方向.

2　分布式数据存储策略

数据一致性是大数据系统中必须解决的问题. 不同的应用场景，对数据一致性要求不同. 根据一致性(consistency)要求的强弱不同，分布式数据存储策略可分为 ACID 和 BASE 两大类.

ACID 是指数据库事务具有的 4 个特性：原子性(atomicity)、一致性(consistency)、隔离性(isolation)、持久性(durability). ACID 中的一致性要求比较强，每个事务的执行务必保证数据库从执行前的一致性状态迁移到执行后的另一个一致性状态. 而 BASE 对一致性要求较弱，它的 3 个特征分别是：基本可用(basically available)、软状态/柔性事务(soft-state，即状态可以有一段时间的不同步)和最终一致性(eventual consistency).

针对 ACID 和 BASE 两种策略，下面重点阐述工业界和学术界几种典型的数据库系统.

2.1　BASE

依据底层架构和支持的数据结构，BASE 可进一步细分为基于键-值对的、基于列的、基于文档的以及基于图的数据库系统.

2.1.1　基于键-值对的数据库系统

基于键-值对的数据库系统的核心思想是，所有数据均以(key，value)的形式存储，其中 key 表示唯一的关键字，value 表示真正的数据，所有关于数据的访问均以关键字 key 进行搜索查询，通常支持 get、set 等操作.

在分布式系统中，横向扩展、数据可靠性以及单点故障给分布式键-值对存储系统带来了困难.

针对这些问题，亚马逊公司研发的 Dynamo[12] 系统，使用基于环的一致性哈希算法来应对计算资源动态加入和退出. 具体来讲，把所有 Dynamo 系统中的节点看作一个环，并且为环上的每个节点分配一个令牌，分配了不同令牌的节点位于环的不同位置，并且任意 2 个令牌是不同的. 然而，在系统中的节点是不对称的，即这些节点可能具有不同的计算能力，若每个节点对应一个令牌，那么会导致较低计算能力节点处理过多数据，而具有丰富计算能力的节点管理数据较少从而浪费了部分计算资源. 为了解决该问题，Dynamo 引入“虚节点”技术，分配给虚节点的令牌实际上由实际节点所持有，从而实际节点可拥有多个令牌. 为了保证数据的可靠性，除了原始数据外，还需将数据拷贝到原始数据所在节点的多个下游节点上. 因此，当读取数据时，若分配到的读取节点失效，那么沿着环的方向寻找可用的存储该数据拷贝的下游节点. Dynamo 的节点发现机制、写操作高可用以及故障处理机制使其成为众多基于键-值对存储系统的基础. Voldemort[13] 是 Dynamo 的一个开源实现，支持数据自动复制、数据自动分割、可插拔策略等.

Cassandra[14] 系统架构与 Dynamo 类似，是基于一致性哈希的完全 P2P 架构，每行数据通过哈希来决定应该存在哪个或哪些节点中. 集群没有 master 的概念，所有节点都是同样的角色，彻底避免了整个系统的单点问题导致的不稳定性，集群间的状态同步通过 Gossip 协议进行 P2P 通信. 每个节点都把数据存储在本地，接受来自客户端的请求. 每次客户端随机选择集群中的一个节点来请求数据，接受请求的节点将对应的 key 在一致性哈希的环上定位到存储这个数据的节点，将请求转发到对应的节点上，并将对应若干节点的查询反馈给客户端.

Cassandra 和 Dynamo 在应对数据的一致性、可用性和分区可容忍性方面同样比较灵活. 具体来讲，Cassandra 的每个 keyspace 可配置一行数据写入多个节点(设这个数为 N)来保证数据不会因为机器宕机或磁盘损坏而丢失，即保证了 CAP 中的分区容忍度.

2.1.2　基于列的数据库系统

BigTable[15] 是 Google 公司研发的最早的基于列的存储系统，该系统支持数据表的横向扩展，用于处理包括 Web 索引、Google 地图等大型系统.

在数据的逻辑组织方面，由于 BigTable 可能要处理高达上百万列的记录，但每个记录实际上存在的列较少，大部分为空值，即列是稀疏的. 实际上，每次读取的数据列往往较少，且部分列同时被请求，因此，BigTable 采用多级映射数据结构：1) 行关键字；2) 列簇和列簇下的列关键字(这些列往往被同时请求)；3) 版本时间戳. 用时间戳来管理多个版本，当版本数量较多时，删除较老的版本.

在分布式体系架构上，BigTable 采用 GFS 分布式文件系统，并设计了 Chubby 分布式锁服务来解决分布式并发、同步和资源分配等问题. BigTable 的节点主要分为 2 类：1) 1 个或多个 Master 节点，用

于处理元数据相关的操作并支持负载均衡;2) 可横向扩展的多个 Tablet 节点,用于存储数据库的 Tablet 区块,所有的数据访问都要映射到相应的 Tablet. 由于 BigTable 采用了列簇和列存储模式,在数据压缩方面具有很大的优势. 为了加速列簇和列的管理,设计了 SSTable 的格式作为 Tablet. 具体的架构如图 3 所示.

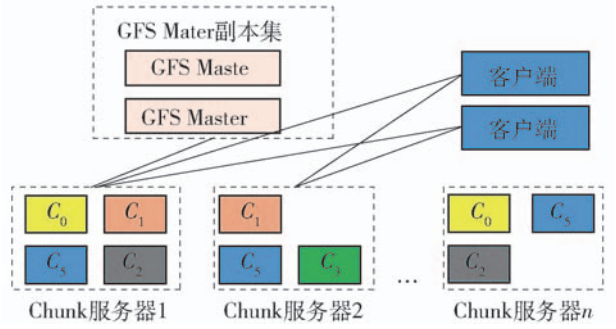

图 3 BigTable 架构

Fig. 3 BigTable architecture

根据 BigTable 的设计思想,开源界研发了基于 JVM 的 HBase[16] 和基于 C + + 的 HyperTable[17]. HBase 基于开源的 GFS 分布式文件系统 HDFS,实现了具有高可靠性、高性能、可伸缩和可实时读写的分布式数据库系统,后来发展成为 Apache 的一个顶级项目;而 HyperTable 则实现了分布式结构化组织并向应用提供类似表访问(类 SQL)的接口,两者均广泛应用于工业界各大公司.

2.1.3 基于文档的存储系统

基于文档的存储系统本质上是一种半结构化数据系统,与传统的数据库从语义上较为接近,但是文档型数据的模式(schema)是不固定的. 此处的"文档"就是一个数据记录,并且这个记录对包含的数据类型和内容进行自我描述,典型的文档包括 XML 文档、HTML 文档以及 JSON 文档,如用 JSON 描述一个人的数据记录,可表示为:{name:"小明", sex:"male", age:10}.

当前,典型的文档型数据库系统包括 MongoDB[18] 和 CouchDB[19]. MongoDB 是一个面向集合的系统,即数据被存储在数据集中,一个数据集称为一个集合(类似于关系数据库中的表格,但是不需要预先定义模式),集合中的文档以 BSON(JSON 的二进制序列)的形式存放,Nytro MegaRAID 技术中的闪存高速缓存算法可帮助 MongoDB 快速识别数据库内大数据集中的热数据,提供一致的性能改进.

CouchDB 与 MongoDB 类似,其中的文档以 JSON 的形式存放. 从 CAP 理论的角度讲,CouchDB 侧重于可用性(A)和分区容忍度(P),而 MongoDB 则侧重于一致性(C)和 P.

2.1.4 基于图的数据系统

前面介绍的 3 种系统,都偏重于描述实体本身,然而,在很多情况下,除了实体本身之外,实体与实体之间关联关系也非常重要,如社交媒体中人与人之间的关系,本体论中本体之间的各种关系等. 为了处理这种关系,基于图的数据库系统应运而生.

典型的基于图的数据系统包括 Neo4j[20] 和 Titan[21].

Neo4j 是一个嵌入式的 Java 持久化引擎,该引擎基于传统磁盘文件系统,与传统数据库的最大差异是并非将记录存储在表中,而是存储在图(网络)中. Neo4j 具有良好的可扩展性,可处理大规模数据,例如,具备在一台机器上处理数十亿节点/关系/属性的图的能力,也可扩展到多台机器并行运行. 相对于关系数据库来说,图数据库能够发挥在处理大量复杂、互连接、低结构化的数据的优势. Neo4j 处理的数据是快速变化的动态数据,适合频繁地查询需求,避免了在关系数据库中耗时的多表连接操作带来的性能瓶颈和性能衰退问题. 通过基于图的数据建模模型,Neo4j 能够以相同的速度遍历节点与边,且遍历速度与数据量大小没有任何关系. 另外,Neo4j 还提供了丰富的图算法、推荐类算法和 OLAP 风格的分析运算算子. 而 Titan 则是一个典型的内存数据库系统,支持包括 HBase、Cassandra 等不同层次的存储层,重点在存储和大规模图处理方面做了优化.

不同数据存储应对的场景不同,键-值对适用于快速索引查询场景;列数据库适用于多版本数据场景;文档数据库更多地应用于数据模式多变的场景;图数据库则大量应用于社交网络和本体论相关应用中.

2.2 ACID

基于 BASE 的数据库系统主要强调可用性和弱一致性,这种系统无法较好地处理全球分布的数据存储的一致性问题. 针对这一问题,Google 研发了一系列存储系统.

鉴于已有大数据管理系统难以提供强一致性的缺陷,谷歌设计了具有高可扩展性和高可用性的 Megastore[22] 存储系统. 该系统基于 Bigtable,能够实现类似 RDBMS 的数据模型,支持细粒度的数据分

区；采用 paxos 实现数据的同步复制，即当更新一个记录时，所有备份节点能够近似实时地同步该更新，确保了所有备份的一致性.

在具体的实践中，Google 发现 Megastore 在吞吐量方面存在缺陷，于是进一步研发了 Spanner[23]. Spanner 是一个全球分布式的同步复制数据库，能够支持高达百万台数据节点，管理上万亿条记录. 该系统所管理的节点能够分布在多达几百个数据中心，具有时间标记的数据的多个版本跨越了多个数据中心，并通过多版本和同步复制实现了外部一致性. 该系统采用模式化的、半关系化的表存储数据，且数据在修改时以提交的时间戳标记不同版本，能确保越旧的数据越容易被垃圾回收，应用可根据需求读取不同版本的数据. 与已有 NoSQL 不同的是，Spanner 支持类似关系数据库那样的通用事务，具有基于 SQL 的查询语言. 总体来讲，Spanner 具有如下重要的特征：1）支持细粒度控制副本，可动态配置数据中心管理副本的范围、数量，降低数据读写延迟. 2）支持读和写的外部一致性，支持在一个时间戳下的跨越数据库的全球一致性的读操作. 这些特征使得 Spanner 具备在全球分布式的一致的备份、一致的 MapReduce 执行和原子模式变更成为可能. CockroachDB[24] 是 Spanner 的一个开源实现.

随着 Google 在广告业务方面的不停扩张，实时性的要求也越来越高，而 BigTable、Megastore 和 Spanner 均无法有效对此进行处理. 因此，Google 又研发了新一代的处理系统 MESA[25]. MESA 是一个具备跨地域复制和近实时特性的可伸缩数据仓库，提供 PB 级数据处理能力和亚秒级响应能力. MESA 提供的数据模型与关系型数据库的数据模型极为相似，以表的方式管理数据，每个表具有特定的模式.

MESA 以多版本方式存储数据，与已有数据更新机制不同的是，在一定时间间隔内，以批处理方式扩散已有更新. 当请求的数据正在更新时返回该数据的前置状态，从而保证数据一致性. 具体来讲，每隔几 min，上游系统以执行一次数据更新的批处理方式扩散已有更新到各副本. 每个独立的、无状态的数据提交者，协调跨全部数据中心的更新操作，为每个更新批处理分配唯一的版本号，并采用 Paxos 一致算法向各节点发布全部与该更新关联的元数据. 当全球范围的大量 MESA 实例合并了一个更新之后，提交者就把该更新的版本号声明为新的提交版本号，并将该值存储在版本数据库里，从而该版本的数据可被请求.

当前，Google 的这三大系统已被全球各大公司所借鉴，国内的 BAT 等也在此基础上研发了各自的内部系统.

分布式数据存储策略是工业界近年来最热的领域之一，各类非结构化数据库层出不穷，然而，由于其应对场景单一，局限性也十分明显，如何对上述各类工具进行融合优化，以适应日益复杂的需求是当前工业界和学术界共同的话题.

3　大数据资源管理

随着大数据高速发展，为解决不同问题，各种计算框架不断涌现，如离线处理框架 MapReduce、迭代式计算框架 Spark 和流处理框架 Storm 等. 不同的框架各有所长，而在实际的生产环境中，常常将这些框架部署在统一的集群中，让它们共享集群的资源，并对资源进行统一管理，即统一资源管理与调度平台，典型的有 Mesos[26] 和 YARN[27].

Mesos 是一个开源的资源管理和调度系统，能够对分布式集群做细粒度资源调度分配. Mesos 诞生于 UC Berkeley，现已成为 Apache 的顶级项目. Mesos 采用两级调度架构，第一级调度是由 Mesos 将资源分配给计算框架，第二级调度是由计算框架自身的调度器给具体 task 分配资源.

Mesos 主要由 Mesos-master、Mesos-slave、Framework 和 Executor 四个组件构成. 其中 Mesos-master 是整个系统的核心，负责管理接入 Mesos 的各个计算框架和 slave，并将 slave 上的资源按照某种策略分配给 Framework. Mesos-slave 将自己的资源量发送给 Mesos-master，并接受 Mesos-master 命令，将资源分配给具体 Framework. Framework 是指外部的计算框架，如 Hadoop、Spark 等，这些框架拥有自身的调度器，用来调度框架内部的 task. Executor 主要用于启动框架内部的 task.

Mesos 具有良好的扩展性、容错性和鲁棒性，而且能做到实时响应，可以适配不同的框架，现已被 Twitter、eBay、Apple 等公司采用.

YARN 是一种通用资源管理系统，它从 Hadoop MapReduce V1 架构发展而来，可为上层应用提供统一的资源管理和调度，使集群实现更好的资源管理利用和数据共享. 在 YARN 出现之前，集成管理被集成在 MPv1 中，存在单点故障、计算框架单一等问题，此外这种架构中 Job Tracker 不能有效地实现对超过千台机器的 MapReduce 任务进行调度. YARN 是为解决这些问题而诞生的，其基本思想是将

JobTracker 的 2 个主要功能:资源管理和作业调度/监控分离. YARN 主要包括 3 个模块:全局的 ResourceManager,节点的 NodeManager,以及若干个针对应用程序的 ApplicationMaster. 其中 ResourceManager 负责管理整个集群的资源,并管理应用程序的资源分配. ApplicationMaster 负责具体应用程序的调度和协调,NodeManager 负责每个节点的维护.

YARN 和 Mesos 都能很好地实现集群资源的调度和管理,支持多个计算框架,并且具有良好的扩展性,均采用 ZooKeeper[28] 进行容错. 但 2 种框架的调度机制不同,YARN 采用静态分区(static partitioning),所有资源均由资源管理器(resource manager)进行细粒度统一调度,直接将资源分配给上层应用的具体任务(task);而 Mesos 采用动态分区(dynamical partitioning),首先将资源粗粒度分配给 Framework,然后由 Framework 细粒度分配资源.

4 大数据计算框架

针对不同的应用场景,大数据计算框架主要包括针对历史静态数据的批处理框架、针对快速流式数据的处理框架、针对交互式计算的处理框架、混合处理框架 Lambda 以及针对图数据的处理框架.

4.1 批处理框架 MapReduce

随着数据的海量增长,高效快速处理数据成为企业成功的关键,如搜索引擎公司的 URL 访问率统计、分布式 grep 操作、分布式排序、倒序索引构建、Web 连接图翻转等应用,集中式处理无法有效应对. 因此,Google 公司于 2004 年发布了 MapReduce[29] 编程模型.

MapReduce 的主要贡献在于提供了简洁强大的分布式、并发程序开发接口,该接口把大尺度的计算自动地并发和分布执行,用户无需考虑进程、线程和数据访问等细粒度问题,也释放了重复性数据通信带来的巨大压力. 其编程模式为:输入一组键-值对—处理该键-值对—产生一组新键-值对作为输出. MapReduce 用 2 个阶段来表达该框架:1) Map 阶段. 调用自定义函数 Map,处理输入的键-值对,并产生新的一组中间的键-值对. 2) Reduce 阶段. 调用定义函数 Reduce 处理上一个 Map 或 Reduce 阶段输出的中间键-值对,以及与这个中间键-值对相关的值集合.

当接受 MapReduce 任务时,MapReduce 框架首先把输入文件分成 M 块,根据 M 块数据在计算节点的位置信息,均衡各节点的计算能力,减少数据移动,分发并调度各个 Map 或 Reduce 任务到合适节点上. 从功能上讲,MapReduce 程序提交的目标程序,是 MapReduce 的主控程序 master,经过任务划分之后,具体的 Map 或 Reduce 任务分发到目标节点执行,目标节点采用 worker 调度、协调各个 Map 或 Reduce 子任务. 当计算节点的某个 worker 获得了由 master 分配的子任务后,结合任务分配时附带的信息构造任务的输入,并执行该任务,若是 Map 任务,则自定义 Map 函数并将产生的新的中间结果 key/value 对暂时缓冲到内存,若内存不足则要写入本地文件系统. 当所有属于该 Map 阶段的子任务完成后,则进行依次 shuffle 操作,把 key 值相同的 key/value 对归总到一起,形成 key 值相同的 key/value 集合. Shuffle 之后的结果被定时刷写到本地硬盘,这些中间结果在本地硬盘的位置信息将被发送回 master 作为下一个 Map 或 Reduce 阶段的输入. 若当前阶段为 Reduce 阶段,那么 Reduce 阶段的 worker 将 key 值相同的上一个阶段的输出作为自定义函数 Reduce 的输入;根据业务需求,该 Reduce 阶段处理数据,并生成新的键-值对,若有其余阶段,那么就作为下一阶段的输入.

MapReduce 提供了良好的容错机制,容错粒度包括 JobTracker、TaskTracker 等级别. 目前,一个集群中通常只有一个 JobTracker,一旦 JobTracker 出现错误往往需要人工介入,但是用户可以通过一些参数进行控制从而让所有作业恢复运行,TaskTracker 的容错则通过心跳检测、黑名单、灰名单机制来对失效的 TaskTracker 节点进行及时处理达到容错效果.

目前,典型的 MapReduce 框架包括 Google 公司未开源的批处理系统以及其开源版本 Hadoop,后者已广泛应用于工业界各大公司.

4.2 流式处理框架 Storm

在信息爆炸的年代,人们对于信息的实时性要求越来越高,如应急监控系统,可实时对多源数据进行监控,一旦发生紧急状况,能实时预警并快速做出响应,如果错过时机,可能造成严重的后果. 因此,需要应急监控系统具有低延迟、高性能、分布式、可扩展、容错等特点. 流式处理框架因此应运而生.

Storm[30] 是 Twitter 的一个开源框架. Storm 是由 BackType 开发的一个分布式的、容错的实时计算系统. 在 Storm 集群中,Spout 组件负责将输入流数据传递给 Bolt 组件,Bolt 组件以用户指定的方式处理数据(持久化或者处理后转发给其他的 Bolt),

Storm 集群可看成一条由 Bolt 组件构成的处理链，每个 Bolt 负责对数据做相应的处理.

Storm 集群包含一个主控节点(master node)和若干个工作节点(work node)，两者的角色和任务分工如下：1) 主控节点接受提交的任务，并负责将该任务分发给工作节点执行. 这是通过运行在主控节点上的后台程序 Nimbus 实现的. Nimbus 同时也负责监控集群内任务的运行状态，当有失败任务时，将失败的人物分发给其余工作节点去执行. 由此可见，它的作用类似于 Hadoop 中 JobTracker 的角色. 2) 工作节点接受来自主控节点分发的任务，并通过在工作节点上运行的后台程序 Supervisor 调度该任务去执行特定的代码. Storm 具有编程模型简单、容错性高、水平扩展方便以及可快速可靠地处理消息等特点. 当前，Storm 已广泛运用于各大 IT 公司的流处理场景.

4.3　交互式计算框架 Spark

谷歌的 MapReduce 和 Hadoop 的出现实现了对大型集群并行数据处理的工具. 但是这些工具也存在一些不足，在不同的计算引擎之间进行资源的动态共享比较困难、迭代式计算性能比较差，只适合批处理，对关联数据的研究和复杂算法的分析效率低下.

基于以上问题，推出了一个全新的统一大数据处理框架 Spark[8, 31]. 为了克服在 Hadoop 平台上处理数据迭代性能差和数据共享困难等问题，Spark 提出了一个新的存储数据概念 RDD(一种新的抽象的弹性数据集). RDD 的本质就是在并行计算的各个阶段进行有效的数据共享. RDD 是只读的、分区记录的集合，是在稳定物理存储中的数据集和其他已有的 RDD 上执行确定性操作来创建的. 这些确定性操作称之为转换，如 map、filter、groupBy、join. RDD 提供内存存储接口，使得数据存储和查询效率比 Hadoop 高很多.

RDD 除了提供内存存储和各种类型并行计算外，还可以自动从故障中恢复，实现了基于 Lineage 的容错机制. RDD 的转换关系，构成了计算链(compute chain)，可以把这个 compute chain 认为是 RDD 之间演化的历史记录. 在部分计算结果丢失时，只需根据这个历史记录重算即可. Lineage 故障恢复不需要复制数据，这样可以在构建 RDD 时节约大量时间.

与 Hadoop 的 MapReduce 相比，Spark 基于内存的运算要快 100 倍以上，而基于磁盘的运算也要快 10 倍以上. Spark 实现了高效的 DAG 引擎，可以通过基于内存来高效处理数据流.

Spark 支持 JAVA、Python 和 Scala 的 API，使得用户可以快速构建不同的应用. Spark 可以用于批处理、交互式查询(通过 Spark SQL)、实时流处理(通过 Spark Streaming)、机器学习(通过 Spark MLlib)和图处理(通过 Spark GraphX). Spark 可以非常方便地与其他的开源产品进行融合.

4.4　Lambda 混合架构

鉴于 MapReduce 批处理模式适合大体量数据，但实时性不足，而 Storm 计算架构具有良好的近实时数据处理能力，但批处理有所欠缺，研究人员提出了支持 2 种架构的 Lambda 混合架构，该混合架构对不同的业务需求进行了良好的平衡.

首先，Lambda 混合架构包括快速数据处理和批处理 2 个层面，当数据到来时，分别将数据送入批处理层和实时处理层. 当数据进入批处理层时，以批量写入模式将原始数据存储到 HDFS 或 S3 中，而不是在有索引情况下的随机写入. 有原始数据的存储避免了系统硬件问题和人为错误带来的损失. 并且，该批处理层依然可以运行传统的 MapReduce 程序，进行数据分析和处理，而分析的结果可用于快速数据处理层中.

在实时数据处理层，拿到批量数据时，能以最快的速度处理和分析该数据，并为用户提供查询. 比如，在某些在线应用中，快速数据处理层利用最新的计算结果，可以处理较为耗时的任务，满足用户的要求，并且该快速数据处理层可以和批处理层共享数据分析的结果. 在批处理得到精确结果之前，即时服务采用实时数据层得到最新结果，在融合了来自批处理层的精确结果之后，实时处理层能够为用户提供更加精确的服务.

在算法方面两者有所区别，数据追加模式的批处理层不能反应最新的数据状态，但可以提供比如数据索引机制，即时的历史数据查询和分析任务，而快速数据处理层，可使用增量式的算法提供最新的服务. 数据提供方式可通过服务层实现.

4.5　图处理框架 Pregel

随着社交网络的深度发展，基于社交网络的各种计算需求如社区发现等越来越多；另外，本体论也逐渐成为各学科研究的一种规范化方法. 图是社交网络和本体论的基本数据结构，图结构上的计算是解决社交网络和本体论中问题的基本途径.

Pregel[32]是 Google 公司研发的一个用于分布式

图计算、基于整体同步并行计算模型(bulk synchronous parallel,BSP)的计算框架,主要用于图遍历(BFS)、最短路径(SSSP)、PageRank 计算等,其主要目标是解决 MapReduce 的低效以及并行图数据库 Parallel BGL 中无容错机制的问题.

Pregel 的基本思想是:以节点为中心进行计算,节点有活跃及不活跃2种状态,初始时,每个节点都处于活跃状态,完成计算后每个节点主动进入不活跃状态,如果接收到信息,则激活,没有活跃节点和消息时,整个算法结束.

Pregel 架构有如下3个主要特征:

1)采用主/从结构来实现整体功能. 主节点负责对整个图结构的任务进行切分,根据节点的 ID 分配到从节点,从节点进行超步计算并返回给主节点.

2)具有良好的容错机制. 通过 Checkpoint 机制实行容错.

3)使用 GFS 和 BigTable 作为持久性存储.

大数据计算框架具有计算引擎专业化、处理平台多样化、数据处理实时化等趋势[33]. 计算框架的复杂性日趋复杂(如 Spark 从最初的不到1万行代码到现在的近百万行),如何根据不同的应用场景对复杂的平台进行优化,以及如何在如此复杂的平台上研发更加高效的分析算法是科研工作者需要解决的难题.

5 结论

1)本文重点对当前大数据文件系统、分布式存储策略、资源调度管理以及大数据计算框架中的典型技术进行了阐述. 这些技术均是为了解决现实应用场景中的问题而发展来的,在工业界赢得了广泛的认同和应用,并且引起了学术界足够的关注.

2)在大数据领域,当前依然存在2个问题亟待解决:① 如何通过适当增加数据语义提高计算效率. 无论是当前火热的 MapReduce,还是 Spark 等技术,均是一种"蛮力型"的计算,单纯通过提高并发度提高计算效率,无法从根本上降低计算的规模和复杂度,是一种较为耗费资源的计算方式. 如果能够通过适当增加数据语义,在计算之前以较低的代价减少计算的规模,则能极大地提高计算效率. ② 时空计算问题. 当前的大数据技术通常没有考虑时空属性,其搜索查询模式一般基于关键字,很少考虑时空计算问题;少量的研究重点关注空间计算或者时间计算,缺乏针对时空计算的大数据系统,而这一系统在物联网蓬勃发展的今天至关重要.

参考文献:

[1] VANCE A. Start-up goes after big data with Hadoop helper [EB/OL]. [2016-12-27]. http://bits.blogs.nytimes.com/2010/04/22/start-up-goes-after-big-data-with-hadoop-helper/?dbk.

[2] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system [C] // Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. New York: ACM, 2003: 29-43.

[3] SHVACHKO K, KUANG H, RADIA S, et al. The Hadoop distributed file system [C] // IEEE 26th Symposium on Mass Storage Systems and Technologies. Washington, DC: IEEE Computer Society, 2010: 1-10.

[4] LI H, GHODSI A, ZAHARIA M, et al. Tachyon: reliable, memory speed storage for cluster computing frameworks[C] // Proceedings of the ACM Symposium on Cloud Computing(SOCC'14). New York: ACM, 2014: 1-15.

[5] HE Y, LEE R, HUAI Y, et al. RCFile: a fast and space-efficient data placement structure in MapReduce-based warehouse systems [C] // Proceedings of the 27th International Conference on Data Engineering. New York: ACM, 2011: 1199-1208.

[6] LE DEM J. The striping and assembly algorithms from the Dremel paper[EB/OL]. [2013-09-12]. https://github.com/Parquet/parquet-mr. wiki. git.

[7] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: a scalable, high-performance distributed file system[C] // The 7th Symposium on Operating Systems Design and Implementation. New York: ACM, 2006: 307-320.

[8] ZAHARIA M. An architecture for fast and general data processing on large clusters[D]. Berkeley: University of California Berkeley, 2014.

[9] THUSOO A, SARMA J S, JAIN N, et al. Hive—a warehousing solution over a map-reduce framework[J]. Proceedings of the VLDB Endowment, 2009, 2(2): 1626-1629.

[10] HUAI Y, CHAUHAN A, GATES A, et al. Major technical advancements in apache hive[C] // Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2014: 1235-1246.

[11] MELNIK S, GUBAREV A, LONG J J, et al. Dremel: interactive analysis of Web-scale datasets [J]. Communications of the ACM, 2011, 54(6): 114-123.

[12] DECANDIA G, HASTORUN D, JAMPANI M, et al. Dynamo: amazon's highly available key-value store[C] // Proceedings of the 21st ACM Symposium on Operating Systems Principles. New York: ACM, 2007: 205-220.

[13] SUMBALY R, KREPS J, GAO L, et al. Serving large-

scale batch computed data with project voldemort[C]// Proceedings of the 10th USENIX Conference on File and Storage Technologies. New York: ACM, 2012: 18-30.

[14] LAKSHMAN A, MALIK P. Cassandra: a decentralized structured storage system[J]. Operating Systems Review, 2010, 44(2): 35-40.

[15] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: a distributed storage system for structured data[J]. ACM Trans Comput Syst, 2008, 4(2): 1-26. DOI: //10.1145/1365815.1365816.

[16] Apache. Apache HBase[Z/OL]. [2014-10-21]. http://hbase.apache.org/.

[17] RIOS G, JUDD D. Load balancing for hypertable[C]// Proceedings of the 8th AAAI Conference on AI for Data Center Management and Cloud Computing (AAAIWS'11-08). San Francisco : AAAI Press, 2011: 24-26.

[18] CHODOROW K, DIROLF M. MongoDB-the definitive guide: powerful and scalable data storage [M]. Cambridge: O'Reilly Media, Inc., 2010: 1-193.

[19] ANDERSON J C, LEHNARDT J, SLATER N. CouchDB—the definitive guide: time to relax [M]. Cambridge: O'Reilly Media, Inc.,2010: 1-245.

[20] WEBBER J. A programmatic introduction to Neo4j[C]// Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH'12). New York: ACM, 2012: 217-218. DOI: //10.1145/2384716.2384777.

[21] ROGGEN D, LOMBRISER C, ROSSI M, et al. Titan: an enabling framework for activity-aware "pervasive apps" in opportunistic personal area networks [J]. EURASIP Journal on Wireless Communications and Networking, 2011(1): 1-22.

[22] BAKER J, BOND C, CORBETT J C, et al. Megastore: providing scalable, highly available storage for interactive services[C]// Fifth Biennial Conference on Innovative Data Systems. New York: ACM, 2011: 223-234.

[23] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally-distributed database [C]// The 10th USENIX Symposium on Operating Systems Design and Implementation. New York: ACM, 2013: 261-264.

[24] 谷雨. 数据存储 CockroachDB 应用技术 Docker 的研究[J]. 信息技术, 2015(7): 187-189.
GU Y. Applied data storage CockroachDB in Docker[J]. Information Technology, 2015 (7): 187-189. (in Chinese)

[25] GUPTA A, YANG F, GOVIG J, et al. Mesa: geo-replicated, near real-time, scalable data warehousing [J]. Proceedings of the VLDB Endowment, 2014, 7(12): 1259-1270.

[26] HINDMAN B, KONWINSKI A, ZAHARIA M, et al. Mesos: a platform for fine-grained resource sharing in the data center [C]// Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation. New York: ACM, 2011: 429-483.

[27] VAVILAPALLI V K, MURTHY A C, DOUGLAS C, et al. Apache Hadoop YARN: yet another resource negotiator[C]// SOCC'13, ACM Symposium on Cloud Computing. New York: ACM, 2013: 1-16.

[28] HUNT P, KONAR M, JUNQUEIRA F P, et al. ZooKeeper: wait-free coordination for Internet-scale systems [C]// 2010 USENIX Annual Technical Conference. New York: ACM, 2010: 653-710.

[29] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Commun ACM, 2008, 51(1): 107-113.

[30] CHRISTENSEN R, WANG L, LI F, et al. STORM: spatio-temporal online reasoning and management of large spatio-temporal data[C]// Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2015: 1111-1116.

[31] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: cluster computing with working sets[C]// The 2nd USENIX Workshop on Hot Topics in Cloud Computing. New York: ACM, 2010: 1765-1773.

[32] MALEWICZ G, AUSTERN M H, BIK A J C, et al. Pregel: a system for large-scale graph processing[C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM, 2010: 135-146.

[33] 程学旗, 靳小龙, 王元卓, 等. 大数据系统和分析技术综述[J]. 软件学报, 2014, 25(9): 1889-1908.
CHENG X Q, JIN X L, WANG Y Z, et al. Survey on big data system and analytic technology[J]. Journal of Software, 2014, 25(9): 1889-1908. (in Chinese)

(责任编辑　梁　洁)